Implementação de TS com BST

- Para definir um nó da árvore, acrescentamos campos key e val aos nós de uma BT.

```
private Node root;

private class Node {
    private Key key;
    private Value val;
    private Node left, right;
    public Node(Key key, Value val) {
        this.key = key;
        this.val = val;
    }
}
```

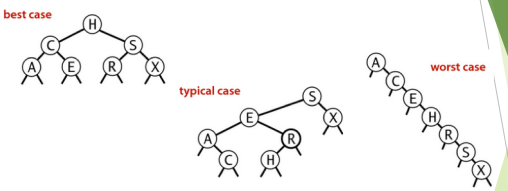
Exemplo de Rastreamento

- Rastreamento da inserção das chaves S E A R C H E X A M P L E, nessa ordem:
- Qualquer busca (get) que termina em um nó x de profundidade *p* visita 1 + *p* nós. O mesmo vale para inserção (put).

Desempenho no Pior Caso

- Toda operação de busca ou inserção visita 1 + *p* nós, sendo *p* a profundidade do último nó visitado.
- Logo, o número de nós visitados não passa de 1 + *h*, sendo *h* a altura da BST.
- (Proposição E) No pior caso, todas as operações sobre uma BST consomem tempo proporcional à altura da árvore.
- Infelizmente, uma BST pode não estar balanceada: sua altura pode estar bem mais perto de *N* que de lg *N*.

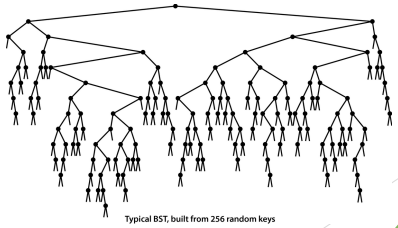
Possibilidades de BST



- Embora BSTs desbalanceadas sejam fáceis de construir (basta inserir as chaves em ordem apoximadamente crescente), elas são *raras* entre as BSTs aleatórias.

Desempenho esperado (= médio)

- BST típica construída com 256 chaves inseridas em ordem aleatória (quantos nós tem essa árvore?):



Typical BST, built from 256 random keys

Desempenho esperado (= médio)

- Analisar desempenho *esperado* de uma implementação é sempre difícil.
- Quantos nós são visitados, em média, durante uma busca em uma BST aleatória com *N* nós?
- Uma *BST aleatória* é uma BST que se obtém inserindo *N* chaves distintas *em ordem aleatória* numa árvore inicialmente vazia.
- Qual a *altura esperada* de uma BST aleatória? Resposta (L. Devroye): aproximadamente 3 lg *N* quando *N* é grande.

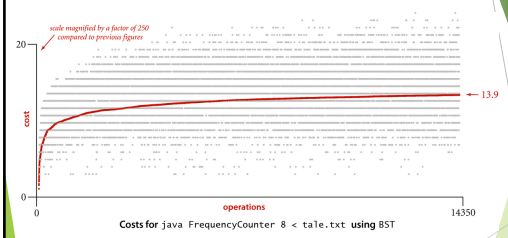
Desempenho esperado (= médio)

- Portanto, o número esperado de nós visitados durante uma busca em uma BST aleatória *não passa de* 3 lg *N*.
- O número *esperado* de nós visitados durante uma busca ou inserção em uma BST aleatória é menor que 3 lg *N*: ele tende a 1.4 lg *N* quando *N* aumenta (veja Proposições C e D) no livro.
- A experiência mostra que o número 1.4 lg *N* também é aproximadamente correto em muitas situações práticas.

Desempenho esperado (= médio)

- Exemplo: SW trocou ST por BST no programa-cliente FrequencyCounter e usou o programa para examinar as palavras com 8 ou mais letras do arquivo tale.txt. O gráfico do próximo slide (copiado da p.405 do livro) mostra o número de nós visitados (= número de comparações feitas) por cada put(). Os pontos vermelhos dão a média corrente:

Desempenho esperado (= médio)



Costs for `java FrequencyCounter 8 < tale.txt` using BST

13.9 não fica longe de 1.4 lg *N*, que vale 17.6

Resumo das 3 implementações de TSs vistas até aqui

	Pior caso (tabela com N chaves)		Caso médio (N chaves inseridas em ordem aleatória)	
	Busca	Inserção	Busca bem-sucedida	Inserção
Busca sequencial em lista ligada	N	N	N/2	N
Busca binária em vetor ordenado	lg N	N	lg N	N/2
Busca binária em BST	N	N	1.4 lg N	1.4 lg N